Cryptography and Network Security
UNIT-2

Syllabus: Block Ciphers & Symmetric Key Cryptography

Traditional Block Cipher Structure, DES, Block Cipher Design Principles, AES-Structure, Transformation functions, Key Expansion, Blowfish, CAST-128, IDEA, Block Cipher Modes of Operations

2.1.Introduction:

A block cipher is an encryption/decryption scheme in which a block of plaintext is treated as a whole and used to produce a ciphertext block of equal length.

Many block ciphers have a Feistel structure. Such a structure consists of a number of identical rounds of processing. In each round, a substitution is performed on one half of the data being processed, followed by a permutation that interchanges the two halves. The original key is expanded so that a different key is used for each round.

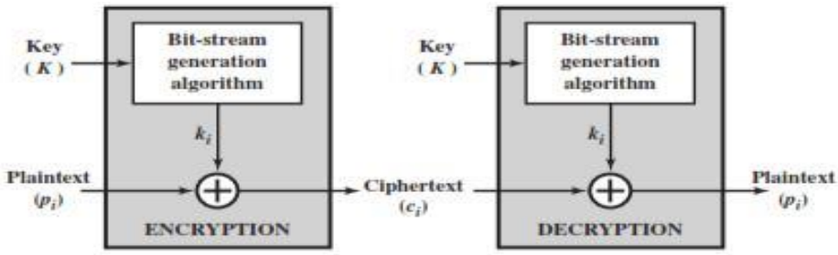
The Data Encryption Standard (DES) has been the most widely used encryption algorithm until recently. It exhibits the classic Feistel structure. DES uses a 64-bit block and a 56-bit key.

2.2. Stream Ciphers and Block Ciphers:

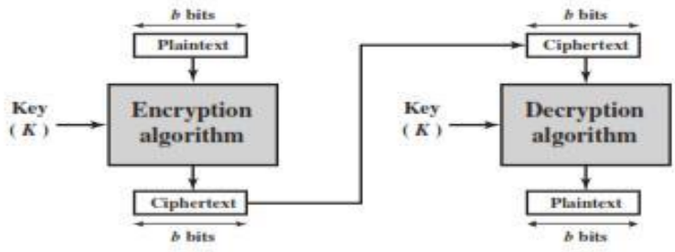
A **stream cipher is one that encrypts a digital data stream one bit or one byte at a time. Examples of** classical stream ciphers are the auto keyed Vigenère cipher and the Vernam cipher.

A **block cipher is** one in which a block of plaintext is treated as a whole and used to produce a ciphertext block of equal length.

Typically, a block size of 64 or 128 bits is used. As with a stream cipher, the two users share a symmetric encryption key



(a) Stream cipher using algorithmic bit-stream generator



(b) Block cipher

2.3.The Feistel Cipher:

Feistel proposed that we can approximate the ideal block cipher by utilizing the concept of a product cipher, which is the execution of two or more simple ciphers in sequence in such a way that the final result or product is cryptographically stronger than any of the component ciphers. The essence of the approach is to develop a block cipher with a key length of k bits and a block length of *n bits, allowing a total of* 2^k *possible transformations, rather than the* $2^n!$ *transformations available with the ideal block cipher.*

In particular, Feistel proposed the use of a cipher that alternates substitutions and permutations, where these terms are defined as follows:

- **Substitution:** Each plaintext element or group of elements is uniquely replaced by a corresponding ciphertext element or group of elements.
- **Permutation:** A sequence of plaintext elements is replaced by a permutation of that sequence. That is, no elements are added or deleted or replaced in the sequence, rather the order in which the elements appear in the sequence is changed.

Feistel's is a practical application of a proposal by Claude Shannon to develop a product cipher that alternates *confusion* and *diffusion* functions

2.3.1 FEISTEL CIPHER STRUCTURE: The left-hand side of Figure depicts the structure proposed by Feistel. The inputs to the encryption algorithm are a plaintext block of length 2w bits and a key. The plaintext block is divided into two halves, L_0 and R_0. The two halves of the data pass through n rounds of processing and then combine to produce the ciphertext block. Each round i has as inputs *Li*-1 and Ri-1 derived from the previous round, as well as a subkey *Ki* derived from the overall K. In general, the subkeys *Ki are different from K and from each other.*

All rounds have the same structure. A **substitution** is performed on the left half of the data. This is done by applying a round function F to the right half of the data and then taking the exclusive-OR of the output of that function and the left half of the data. The round function has the same general structure for each round but is parameterized by the round subkey Ki.

Permutation is performed that consists of the interchange of the two halves of the data. This structure is a particular form of the substitution-permutation network (SPN) proposed by Shannon.

The exact realization of a Feistel network depends on the choice of the following parameters and design features:

Block size: Larger block sizes mean greater security (all other things being equal) but reduced encryption/decryption speed for a given algorithm. The greater security is achieved by greater diffusion. Traditionally, a block size of 64 bits has been considered a reasonable tradeoff and was nearly universal in block cipher design. However, the new AES uses a 128-bit block size.

Key size: Larger key size means greater security but may decrease encryption/ decryption speed. The greater security is achieved by greater resistance to brute -force attacks and greater confusion. Key sizes of 64 bits or less are now widely considered to be inadequate, and 128 bits has become a common size.

Number of rounds: The essence of the Feistel cipher is that a single round offers inadequate security but that multiple rounds offer increasing security. A typical size is 16 rounds.

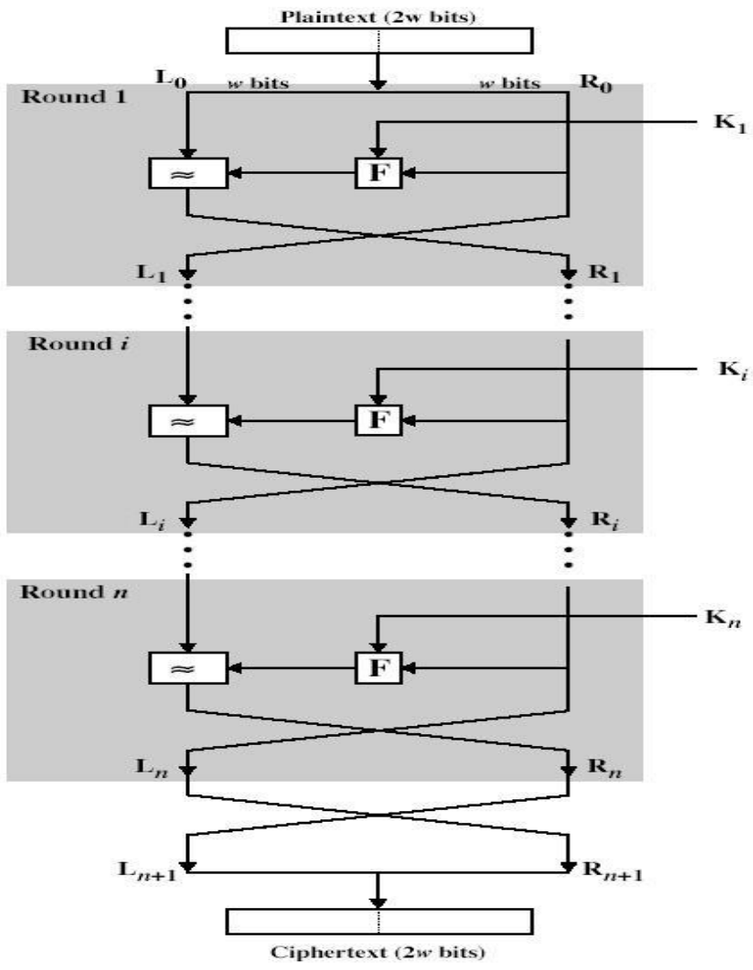


Fig: Feistel Cipher structures

Subkey generation algorithm: Greater complexity in this algorithm should lead to greater difficulty of cryptanalysis.

Round function F: Again, greater complexity generally means greater resistance to cryptanalysis.

There are two other considerations in the design of a Feistel cipher:

Fast software encryption/decryption: In many cases, encryption is embedded in applications or utility functions in such a way as to preclude a hardware implementation. Accordingly, the speed of execution of the algorithm becomes a concern.

Ease of analysis: Although we would like to make our algorithm as difficult as possible to cryptanalyze, there is great benefit in making the algorithm easy to analyze. That is, if the algorithm can be concisely and clearly explained, it is easier to analyze that algorithm for cryptanalytic vulnerabilities and therefore develop a higher level of assurance as to its strength. DES, for example, does not have an easily analyzed functionality.

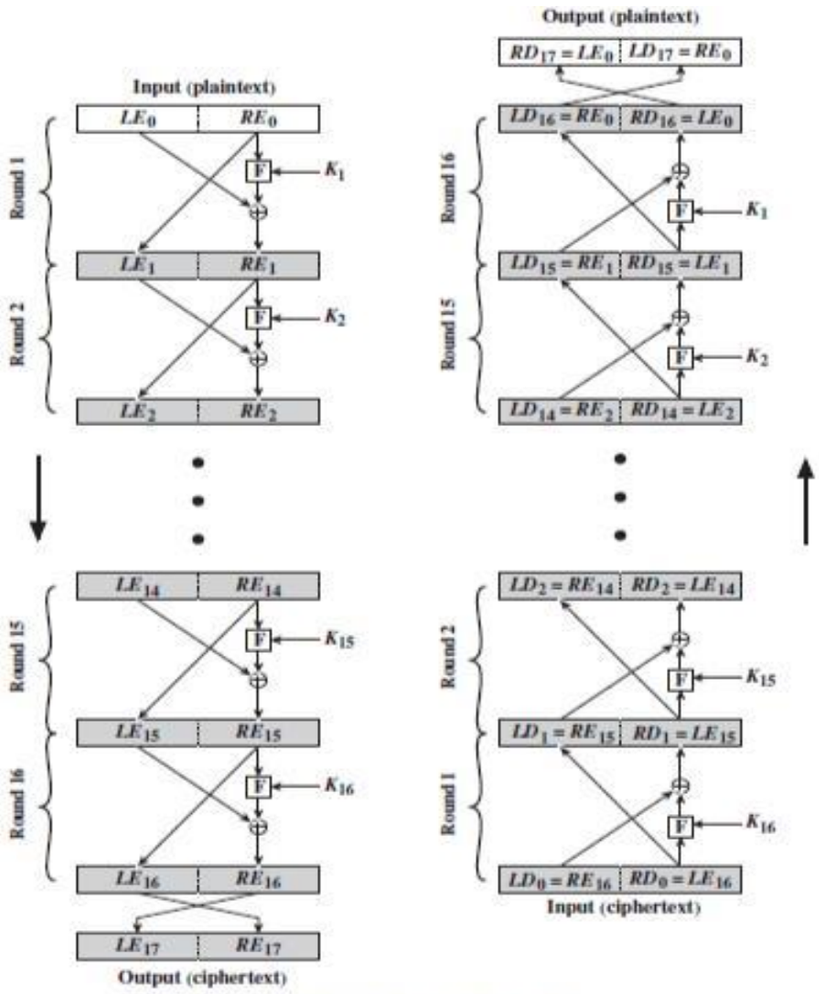


Figure 3.3 Feistel Encryption and Decryption (16 rounds)

2.3.2. Feistel Decryption Algorithm:

The process of decryption with a Feistel cipher is essentially the same as the encryption process.

The rule is as follows:

Use the ciphertext as input to the algorithm, but use the subkeys *K* in reverse order.

That is, use K_n in the first round, K_{n-1} in the second round, and so on until *K* is used in the last round. This is a nice feature because it means we need not implement two different algorithms, one for encryption and one for decryption.

To see that the same algorithm with a reversed key order produces the correct result, which shows the encryption process going down the left-hand side and the decryption process going up the right-hand side for a 16-round algorithm.

For clarity, we use the notation LE_i and RE_i for data traveling through the encryption algorithm and LD_i and RD_i for data traveling through the decryption algorithm.

The diagram indicates that, at every round, the intermediate value of the decryption process is equal to the corresponding value of the encryption process with the two halves of the value swapped.

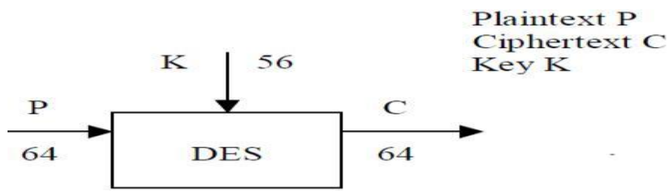
After the last iteration of the encryption process, the two halves of the output are swapped, so that the ciphertext is $RE_{16}||LE_{16}$. The output of that round is the ciphertext. Now take that ciphertext and use it as input to the same algorithm. The input to the first round is $RE_{16}||LE_{16}$, which is equal to the 32-bit swap of the output of the sixteenth round of the encryption process.

If you clearly observe that the output of the first round of the decryption process is equal to a 32-bit swap of the input to the sixteenth round of the encryption process. First, consider the encryption process.

encryption process	On the decryption side,
$LE_{16} = RE_{15}$ $RE_{16} = LE_{15} \oplus \mathrm{F}(RE_{15}, K_{16})$ **in general** $LE_i = RE_{i-1}$ $RE_i = LE_{i-1} \oplus \mathrm{F}(RE_{i-1}, K_i)$	$LD_1 = RD_0 = LE_{16} = RE_{15}$ $RD_1 = LD_0 \oplus \mathrm{F}(RD_0, K_{16})$ $= RE_{16} \oplus \mathrm{F}(RE_{15}, K_{16})$ $= [LE_{15} \oplus \mathrm{F}(RE_{15}, K_{16})] \oplus \mathrm{F}(RE_{15}, K_{16})$

2.4. Data Encryption Standard:

- DES is a Symmetric-key algorithm for the encryption of electronic data.
- DES originated at IBM in 1977 & was adopted by the U.S Department of Defence. Now it is under the NIST (National Institute of Standard & Technology)
- Data Encryption Standard (DES) is a widely-used method of data encryption using a private (secret) key
- DES applies a 56-bit key to each 64-bit block of data. The process can run in several modes and involves 16 rounds or operations.



Inner workings of DES:

DES (and most of the other major symmetric ciphers) is based on a cipher known as the Feistel block cipher. This was a block cipher developed by the IBM cryptography researcher Horst Feistel in the early 70's. It consists of a number of rounds where each round contains bit-shuffling, non-linear substitutions (S-boxes) and exclusive OR operations. Most symmetric encryption schemes today are based on this structure (known as a Feistel network).

Overall structure

DES (and most of the other major symmetric ciphers) is based on a cipher known as the Feistel block cipher.

Looking at the left-hand side of the figure, we can see that the processing of the plaintext proceeds in three phases.

- First, the 64-bit plaintext passes through an initial permutation (IP) that rearranges the bits to produce the *permuted input*.
- This is followed by a phase consisting of sixteen rounds of the same function, which involves both permutation and substitution functions. The output of the last (sixteenth) round consists of 64 bits that are a function of the input plaintext and the key. The left and right halves of the output are swapped to produce the **preoutput**.

- Finally, the preoutput is passed through a permutation that is the inverse of the initial permutation function, to produce the 64-bit cipher text. With the exception of the initial and final permutations, DES has the exact structure of a Feistel cipher,

The right-hand portion of below shows the way in which the 56-bit key is used. Initially, the key is passed through a permutation function. Then, for each of the sixteen rounds, a *subkey* (*Ki*) is produced by the combination of a left circular shift and a permutation. The permutation function is the same for each round, but a different subkey is produced because of the repeated shifts of the key bits.

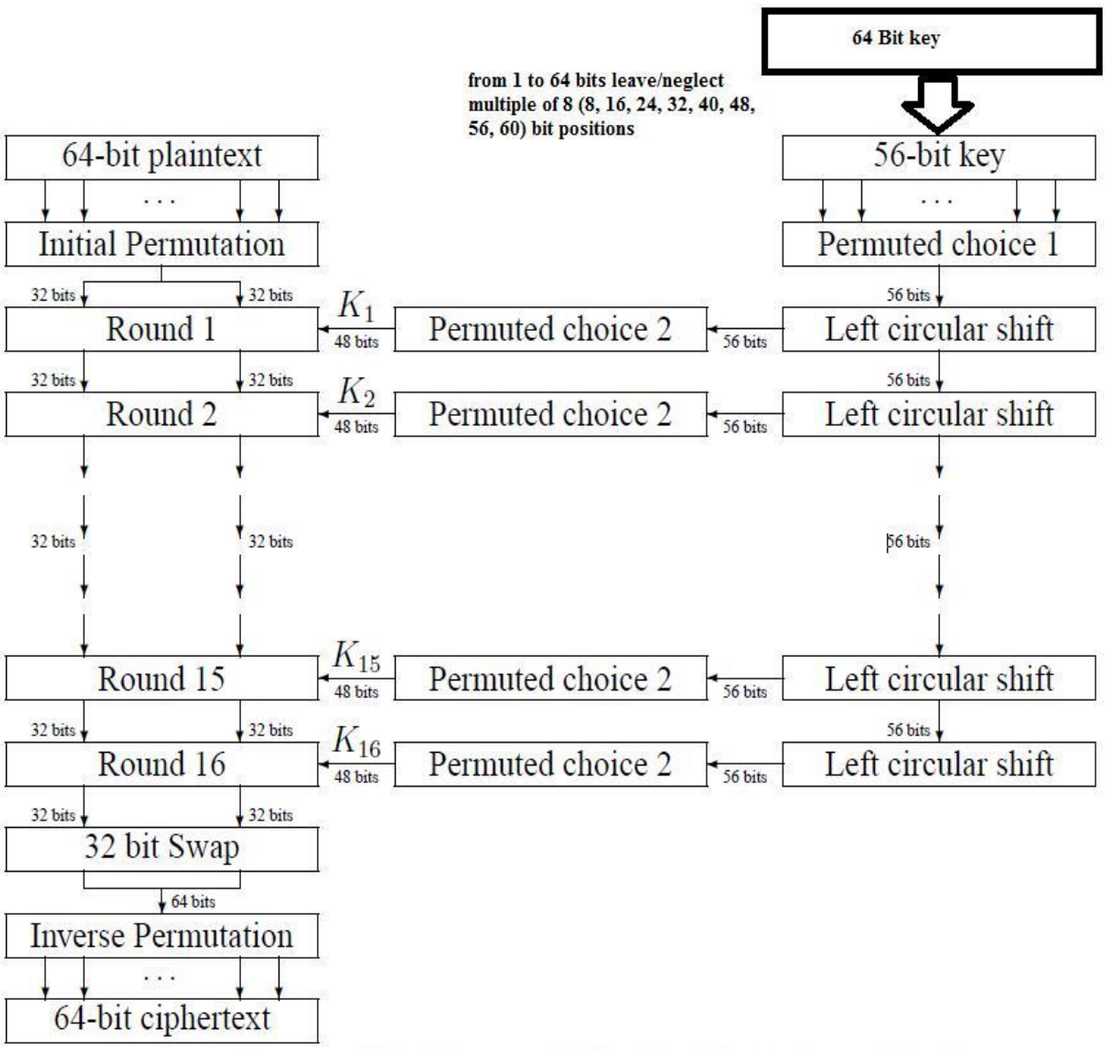


Figure : Flow Diagram of DES algorithm for encrypting data.

Initial Permutation: The initial permutation and its inverse are defined by tables, as shown in Tables (a) and (b), respectively. The tables are to be interpreted as follows. The input to a table consists of 64 bits numbered from 1 to 64. The 64 entries in the permutation table contain a permutation of the numbers from 1 to 64. Each entry in the permutation table indicates the position of a numbered input bit in the output, which also consists of 64 bits.

Table 3.2 Permutation Tables for DES

(a) Initial Permutation (IP)

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

(b) Inverse Initial Permutation (IP^{-1})

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

(c) Expansion Permutation (E)

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

(d) Permutation Function (P)

16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

To see that these two permutation functions are indeed the inverse of each other, consider the following 64-bit input M:

M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8
M_9	M_{10}	M_{11}	M_{12}	M_{13}	M_{14}	M_{15}	M_{16}
M_{17}	M_{18}	M_{19}	M_{20}	M_{21}	M_{22}	M_{23}	M_{24}
M_{25}	M_{26}	M_{27}	M_{28}	M_{29}	M_{30}	M_{31}	M_{32}
M_{33}	M_{34}	M_{35}	M_{36}	M_{37}	M_{38}	M_{39}	M_{40}
M_{41}	M_{42}	M_{43}	M_{44}	M_{45}	M_{46}	M_{47}	M_{48}
M_{49}	M_{50}	M_{51}	M_{52}	M_{53}	M_{54}	M_{55}	M_{56}
M_{57}	M_{58}	M_{59}	M_{60}	M_{61}	M_{62}	M_{63}	M_{64}

Where M_i is a binary digit. Then the permutation $X = (\text{IP}(M))$ is as follows:

M_{58}	M_{50}	M_{42}	M_{34}	M_{26}	M_{18}	M_{10}	M_2
M_{60}	M_{52}	M_{44}	M_{36}	M_{28}	M_{20}	M_{12}	M_4
M_{62}	M_{54}	M_{46}	M_{38}	M_{30}	M_{22}	M_{14}	M_6
M_{64}	M_{56}	M_{48}	M_{40}	M_{32}	M_{24}	M_{16}	M_8
M_{57}	M_{49}	M_{41}	M_{33}	M_{25}	M_{17}	M_9	M_1
M_{59}	M_{51}	M_{43}	M_{35}	M_{27}	M_{19}	M_{11}	M_3
M_{61}	M_{53}	M_{45}	M_{37}	M_{29}	M_{21}	M_{13}	M_5
M_{63}	M_{55}	M_{47}	M_{39}	M_{31}	M_{23}	M_{15}	M_7

If we then take the inverse permutation $Y = \text{IP}^{-1}(X) = \text{IP}^{-1}(\text{IP}(M))$, it can be seen that the original ordering of the bits is restored.

DETAILS OF SINGLE ROUND

Below figure shows the internal structure of a single round. Again, begin by focusing on the left-hand side of the diagram. The left and right halves of each 64-bit intermediate value are treated as separate 32-bit quantities, labeled L (left) and R (right). As in any classic Feistel cipher, the overall processing at each round can be summarized in the following formulas:

$$L_i = R_{i-1}$$
$$R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$$

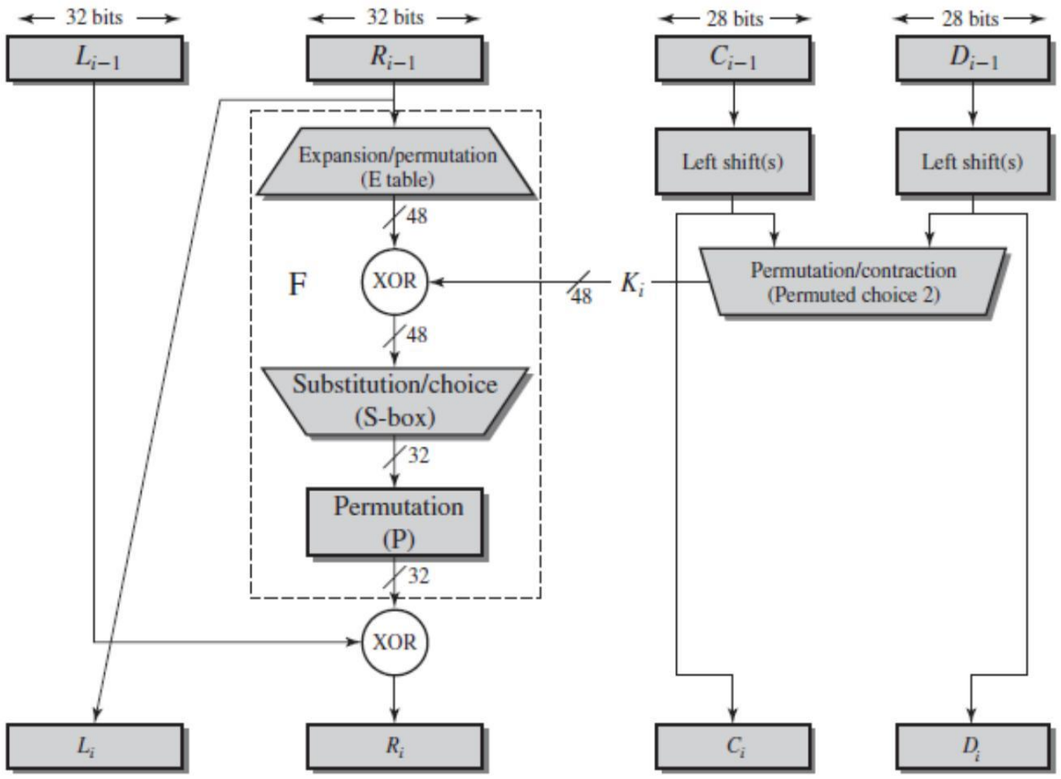


Figure :Single Round of DES Algorithm

The round key K_i is 48 bits. The *R* input is 32 bits. This *R* input is first expanded to 48 bits by using a table that defines a permutation plus an expansion that involves duplication of 16 of the *R* bits (Table 3.2c). The resulting 48 bits are XORed with K_i . This 48-bit result passes through a substitution function that produces a 32-bit output, which is permuted as defined by Table (d). The role of the S-boxes in the function F is illustrated in Figure 3.7. The substitution consists of a set of eight S-boxes, each of which accepts 6 bits as input and produces 4 bits as output. These transformations are defined in Table 3.3, which is interpreted as follows: The first and last bits of the input to box Si form a 2-bit binary number to select one of four substitutions defined by the four rows in the table for. The middle four bits select one of the sixteen columns. The decimal value in the cell selected by the row and column is then converted to its 4-bit representation to produce the output.

For example, in S_1, for input 011001, the row is 01 (row 1) and the column is 1100 (column 12). The value in row 1, column 12 is 9, so the output is 1001. Each row of an S-box defines a general reversible substitution. Figure 3.2 may be useful in understanding the mapping. The figure shows the substitution for row 0 of box S_1. The operation of the S-boxes is worth further comment. Ignore for the moment the contribution of the key (*Ki*). If you examine the expansion table, you see that the 32 bits of input are split into groups of 4 bits and then become groups of 6 bits by taking the outer bits from the two adjacent groups. For example, if part of the input word is

... efgh ijkl mnop ...

This becomes ... defghi hijklm lmnopq ...

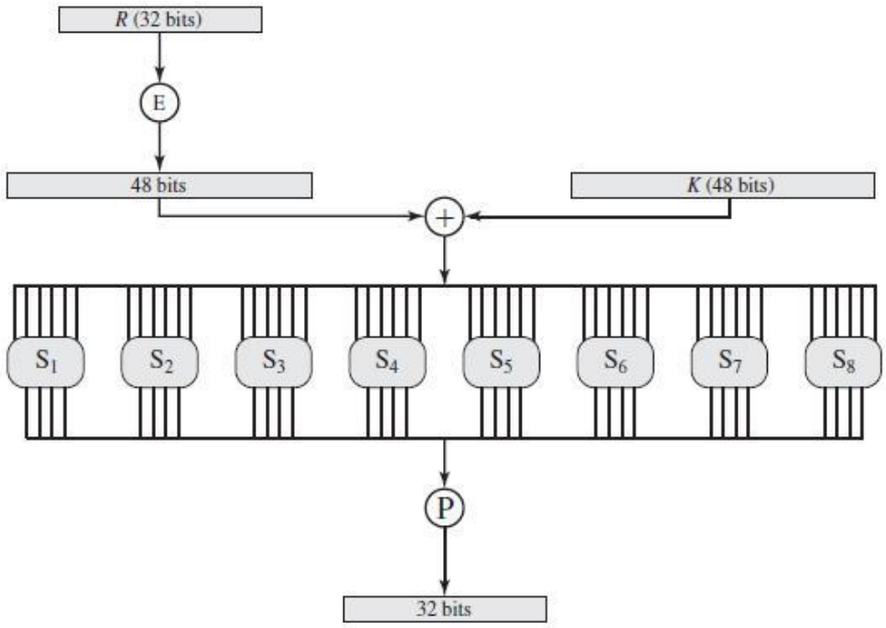


Figure 3.7 Calculation of F(R, K)

The outer two bits of each group select one of four possible substitutions (one row of an S-box). Then a 4-bit output value is substituted for the particular 4-bit input (the middle four input bits). The 32-bit output from the eight S-boxes is then permuted, so that on the next round, the output from each S -box immediately affects as many others as possible.

Substitution Boxes S: Have eight S-boxes which map 6 to 4 bits. Each S-box is actually 4 little 4 bit boxes. Outer bits 1 & 6 (**row** bits) select one rows. inner bits 2-5 (**col** bits) are substituted. Result is 8 lots of 4 bits, or 32 bits. Row selection depends on both data & key

KEY GENERATION:

Returning to above all figures, we see that a 64 - bit key is used as input to the algorithm. The bits of the key are numbered from 1 through 64; every eighth bit is ignored, as indicated by the lack of shading in Table 3.4a. The key is first subjected to a permutation governed by a table labeled Permuted Choice One (Table 3.4b)

The resulting 56-bit key is then treated as two 28-bit quantities, labelled C0 and D_0. At each round, C_{i-1} and D_{i-1} are separately subjected to a circular left shift or (rotation) of 1 or 2 bits, as governed by Table 3.4d. These shifted values serve as input to the next round. They also serve as input to the part l abeled Permuted Choice Two (Table 3.4c), which produces a 48-bit output that serves as input to the Function $F(R_{i-1}, K_i)$.

DES DECRYPTION:

Whatever process we following in the encryption that process is used for decryption also but the order of key is changed on input message (cipher text).

Reverse order of keys are K_{16}, K_{15} ,……, K_1.

The Avalanche Effect:

A desirable property of any encryption algorithm is that a small change in either the plaintext or the key should produce a significant change in the ciphertext.

In particular, a change in one bit of the plaintext or one bit of the key should produce a change in many bits of the ciphertext.

This is referred to as the avalanche effect.

THE STRENGTH OF DES:

The Use of 56-Bit Keys:

With a key length of 56 bits, there are 2^{56} possible keys, which is approximately 7.2×10^{16}. A brute-force attack appears impractical. Assuming that, on average, half the key space has to be searched, a single machine performing one DES encryption per microsecond would take more than a thousand years to break the cipher. Diffie and Hellman postulated that the technology existed to build a parallel machine with 1 million encryption devices, each of which

could perform one encryption per microsecond. This would bring the average search time down to about 10 hours.

The Nature of the DES Algorithm:

Possibilities of cryptanalysis is done by finding the characteristics of DES algorithm. Learning of S-Box logic is complex.

Weakness of the S-boxes not been discovered.

Timing Attacks:

A timing attack is one in which information about the key or the plaintext is obtained by observing how long it takes a given implementation to perform decryptions on various ciphertexts.

A timing attack exploits the fact that an encryption or decryption algorithm often takes slightly different amounts of time on different inputs.

DES appears to be fairly resistant to a successful timing attack.

2.5. Block Cipher Design Principles:

There are three critical aspects of block cipher design: **the number of rounds, design of the function F, and key scheduling**.

Number of Rounds:

The greater the number of rounds, the more difficult it is to perform cryptanalysis, even for a relatively weak F.

In general, the criterion should be that the number of rounds is chosen so that known cryptanalytic efforts require greater effort than a simple brute-force key search attack. This criterion was certainly used in the design of DES.

Design of Function F:

The heart of a Feistel block cipher is the function F, which provides the element of confusion in a Feistel cipher. Thus, it must be difficult to "unscramble" the substitution performed by F.

F must be nonlinear. The more nonlinear F, the more difficult any type of cryptanalysis will be.

Key Schedule Algorithm:

With any Feistel block cipher, the key is used to generate one subkey for each round.

In general, we would like to select subkeys to maximize the difficulty of deducing individual subkeys and the difficulty of working back to the main key.

2.6.Triple DES (3DES):

Triple DES is simply another mode of DES operation. It takes three 64-bit keys, for an overall key length of 192 bits.

The Triple DES then breaks the user provided key into three subkeys, padding the keys if necessary so they are each 64 bits long.

The procedure for encryption is exactly the same as regular DES, but it is repeated three times. Hence the name Triple DES. The data is encrypted with the first key, decrypted with the second key, and finally encrypted again with the third key.

2.7. ADVANCED ENCRYPTION STANDARD(AES):

- The Advanced Encryption Standard (AES) was published by the National Institute of Standards and Technology (NIST) in 2001.
- AES is a block cipher intended to replace DES for commercial applications.
- It uses a 128-bit block size and a key size of 128, 192, or 256 bits.

- AES does not use a Feistel structure. Instead, each full round consists of four separate functions: byte substitution, permutation, arithmetic operations over a finite field, and XOR with a key.

Rijndael was designed to have the following characteristics:

- Resistance against all known attacks
- Speed and code compactness on a wide range of platforms
- Design simplicity

AES parameters:

Key size(words/bytes/bits)	4/16/128	6/24/192	8/32/256
Plaintext block Size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Number of rounds	10	12	14
Round Key size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Expanded key size (words/bytes)	44/176	52/208	60/240

Inner Workings of a Round

The algorithm begins with an Add round key stage followed by 9 rounds of four stages and a tenth round of three stages. This applies for both encryption and decryption with the exception that each stage of a round the decryption algorithm is the inverse of its counterpart in the encryption algorithm. The four stages are as follows:

1. Substitute bytes
2. Shift rows
3. Mix Columns
4. Add Round Key

The tenth round simply leaves out the Mix Columns stage. The first nine rounds of the decryption algorithm consist of the following:

1. Inverse Shift rows
2. Inverse Substitute bytes
3. Inverse Add Round Key
4. Inverse Mix Columns

Again, the tenth round simply leaves out the **Inverse Mix Columns** stage. Each of these stages will now be considered in more detail.

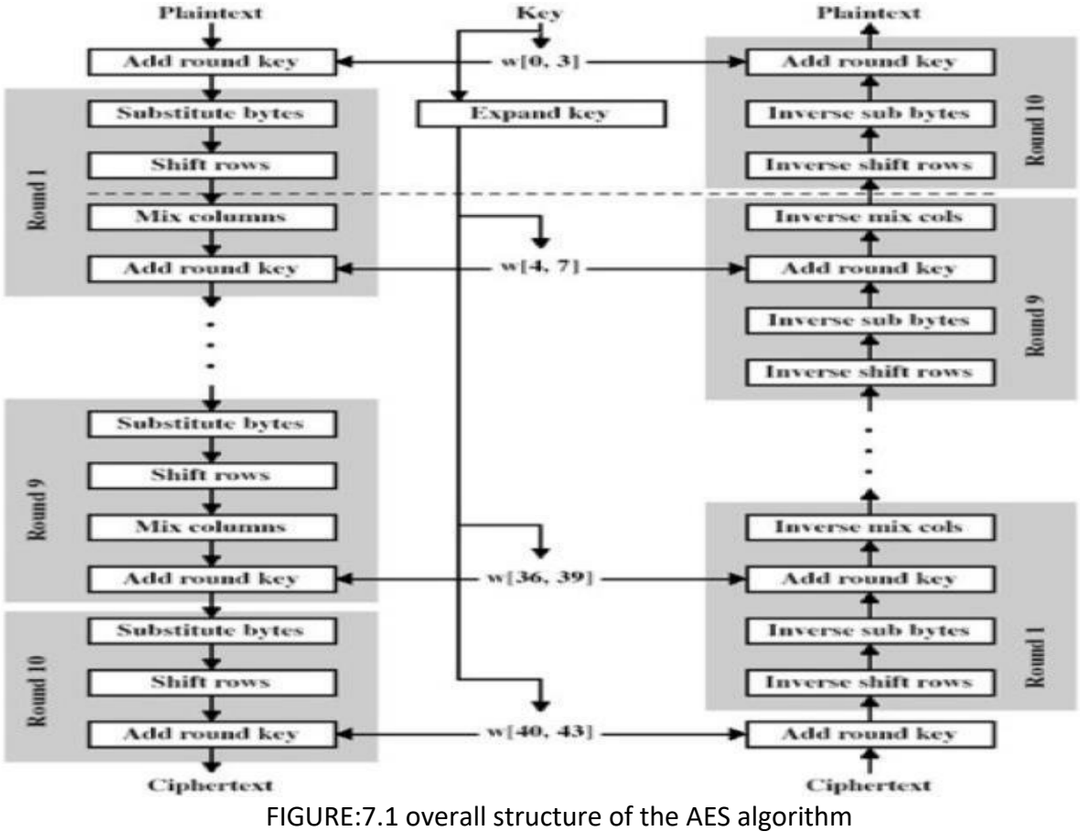


FIGURE:7.1 overall structure of the AES algorithm

Substitute Bytes

This stage (known as SubBytes) is simply a table lookup using a 16×16 matrix of byte values called an s-box. This matrix consists of all the possible combinations of an 8-bit sequence (2^8 = 16 × 16 = 256). However, the s-box is not just a random permutation of these values and there is a well-defined method for creating the s-box tables. The designers of Rijndael showed how this was done unlike the s-boxes in DES for which no rationale was given.

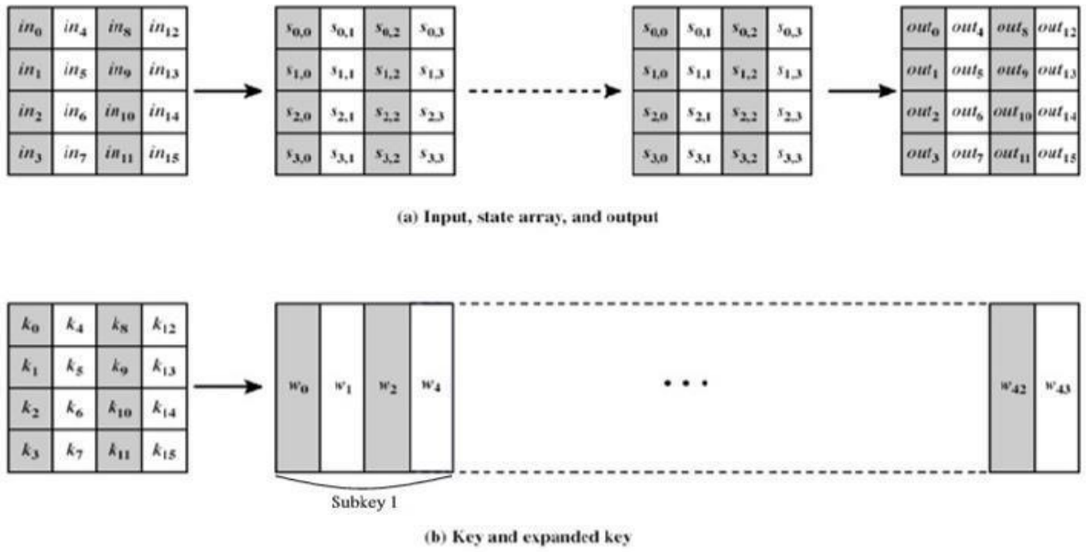


Figure 7.2: Data structures in the AES algorithm.

Again the matrix that gets operated upon throughout the encryption is known as **state**. We will be concerned with how this matrix is effected in each round. For this particular round each byte is mapped into a new byte in the following way: the leftmost nibble of the byte is used to specify a particular row of the s-box and the rightmost nibble specifies a column. For example, the byte {95} (curly brackets represent hex values in FIPS PUB 197) selects row 9 column 5 which turns out to contain the value {2A}.
This is then used to update the **state** matrix. Figure 7.3 depicts this idea.

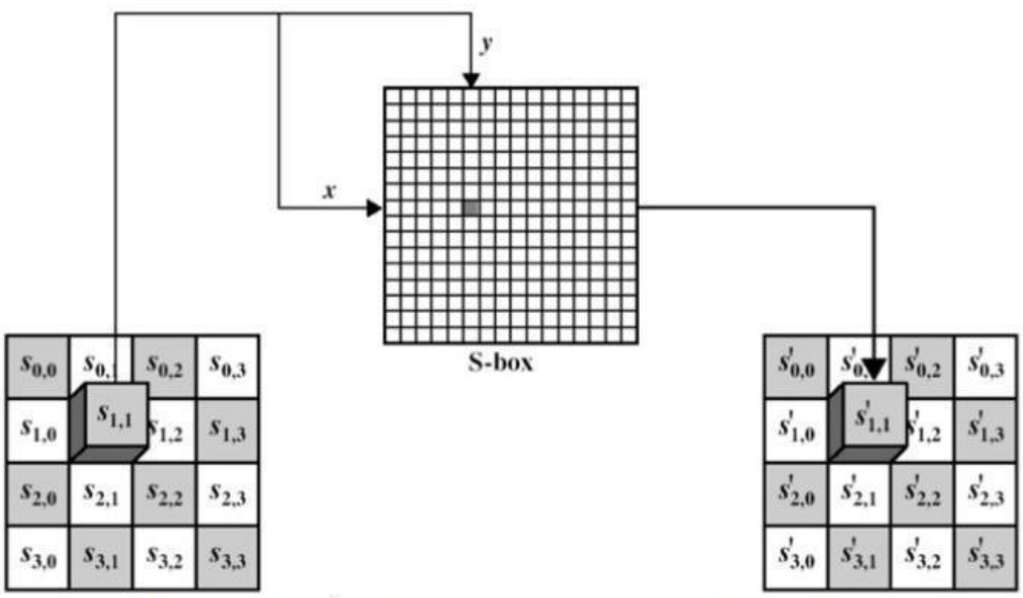


Figure 7.3: Substitute Bytes Stage of the AES algorithm.

The Inverse substitute byte transformation makes use of an inverse s-box. In this case what is desired is to select the value {2A} and get the value {95}. Table 7.4 shows the two s-boxes and it can be verified that this is in fact the case.

The s-box is designed to be resistant to known cryptanalytic attacks. Specifically, the Rijndael developers sought a design that has a low correlation between input bits and output bits, and the property that the output cannot be described as a simple mathematical function of the input.

In addition, the s-box has no fixed points (s-box(a) = a) and no opposite fixed points (s-box(a) = $\bar{a}$) where $\bar{a}$ is the bitwise compliment of a.

Shift Rows Transformation:

Shift row transformation are two types.

Forward Shift row transformation which is used in encryption.

Inverse Shift row transformation which is used in decryption.

FORWARD SHIFT ROW TRANSFORMATION:

- The first row of State matrix is not altered.
- For the second row, a 1-byte circular left shift is performed.
- For the third row, a 2-byte circular left shift is performed.
- For the fourth row, a 3-byte circular left shift is performed. The following is an example of ShiftRows:

87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

→

87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

INVERSE SHIFT ROWS:

- Performs the circular shifts in the opposite direction for each of the last three rows, with a one-byte circular right shift for the second row and soon.

MIX COLUMNS TRANSFORMATION:

Mix columns transformation are two types.

Forward Mix columns transformation which is used in encryption.

Inverse Mix columns transformation which is used in decryption.

Forward Mix columns transformation:

Forward Mix columns transformation called mix columns, operates on each column individually. Each byte of a column is mapped into a new value that is a function of all 4 bytes in that column. The transformation can be defined by the following matrix multiplication on state.

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

inverse Mix columns transformation:

The inverse mix column transformation, called InvMixColumns, is defined by the following matrix multiplication:

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

AddRoundKey Transformation:

- In the forward add round key transformation, called AddRoundKey, the 128 bits of State are bitwise XORed with the 128 bits of the round key.
- The inverse add round key transformation is identical to the forward add round key transformation, because the XOR operation is its own inverse.

47	40	A3	4C
37	D4	70	9F
94	E4	3A	42
ED	A5	A6	BC

$\oplus$

AC	19	28	57
77	FA	D1	5C
66	DC	29	00
F3	21	41	6A

=

EB	59	8B	1B
40	2E	A1	C3
F2	38	13	42
1E	84	E7	D2

AES Key Expansion:

The 128-bit key value can be expanded into 44 words i.e.
44X32=1408bits In each round 4 words will be used i.e. 4x32=128 bits

In Addroundkey first 4 words w0,w1,w2,w3 are
used. In first round,w4,w5,w6,w7 are used and soon.

The 128 bit key is expanded as follows

- First 128 bit key is arranged as a 4x4 matrix each value size is 8-bits
- The first 32 bits (k0,k1,k2,k3) is considered as w0.
- The first 32 bits (k4,k5,k6,k7) is considered as w1.
- The first 32 bits (k8,k9,k10,k11) is considered as w2.
- The first 32 bits (k12,k13,k14,k15) is considered as w4.
- Next 4 words w4,w5,w6,w7 are followed as

w4=w0 ⊕ w3
w5=w1 ⊕ w4
w6=w2 ⊕w5
w7=w3 ⊕w6

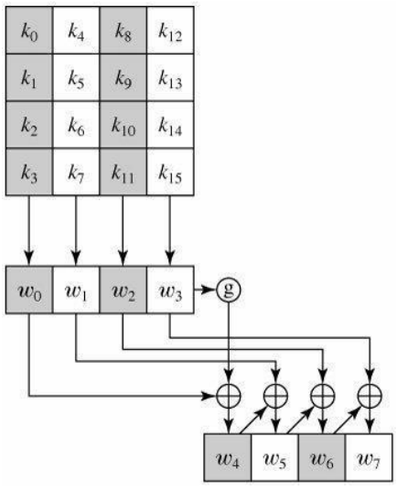


Figure. AES Key Expansion

2.8. BLOWFISH:

- Blow fish is a symmetric block cipher developed by bruce schner in year 1993.
- Blow fish is designed to have following characteristics
  - ✓ Speed: Blowfish encrypts data on 32-bit microprocessor at a rate of 18 clock cycles per byte.
  - ✓ Compact: it can run in less than 5k memory.
  - ✓ Simple: very easy to implements.
  - ✓ Variably secure: the key length is variable and can be as long as 448 bits. This allows a trade-off between higher speed and higher security.
- Blowfish is a Feistel type model.

BLOWFISH ALGORITHM:

- Blowfish is Feistel type model, iterating a simple encryption function 16 times.
- Blowfish block size is 64 & key can be upto 448 bits.
- Blow fish encryption 64bits blocks of plaintext into 64-bit block of cipher.
- Blow fish make use of a key that ranges from 32bits to 448 bits (one to fourteen 32 bit keys).
  - The keys are stored in a k-array (one to 14 32 bits) K_1,K_2---
    -K_j where $1 \leq j \leq 14$.

- That key is used to generate 18 “32 bit” subkeys & four “8*32”bits S-boxes.
  - The subkeys are stored in the p-array

    $P_1, P_2, \text{-------} P_{18}$

There are four s-boxes(each s-box size is 8*32 bits) each with 256 32bit entries.

$S_{1,0}, S_{1,1}, \text{----------} S_{1,255}$
$S_{2,0}, S_{2,1}, \text{----------} S_{2,255}$
$S_{3,0}, S_{3,1}, \text{----------} S_{3,255}$
$S_{4,0}, S_{4,1}, \text{----------} S_{4,255}$

The steps in generating the P-array & S-boxes as follows.

Step1 → initialize first the P-array and then 4 s-boxes in order using the bits of fractional part of the constant π.

Step 2 → Perform a bitwise xor of the P-array & k-array, reusing words from the k-array as needed.

Example $P_1=P_1 \oplus K_1$, $P_1=P_2 \oplus K_2$, -------- $P_{14}=P_{14} \oplus K_{14}$,

$P_{15}=P_{15} \oplus K_1$, $P_{16}=P_1 \oplus K_2$, $P_{17}=P_1 \oplus K_3$, $P_{18}=P_1 \oplus K_4$,

Step 3 → Encrypt the 64 bit block of all zeros using the current P & S-arrays, Replace P1&P2 with the output of the encryption.

$$P_1, P_2 = E_{P,S}[0]$$

Step 4 → Encrypt the output of step 3 using the current P- and S-arrays and replace ***P_3,*** and ***P_4,*** with the resulting ciphertext.

$$P_3, P_4 = E_{P,S}[P_1 \| P_2]$$
$$\cdots$$
$$P_{17}, P_{18} = E_{P,S}[P_{15} \| P_{16}]$$

Step 5 → Continue this process to update all elements of P and then, in order, all elements of S, using at each step the output of the continuously changing Blowfish algorithm.

$$S_{1,0}, S_{1,1} = E_{P,S}[P_{17} \| P_{18}]$$
$$\cdots$$
$$S_{4,254}, S_{4,255} = E_{P,S}[S_{4,252} \| S_{4,253}]$$

The update process can be summarized as follows

$$P_1, P_2 = E_{P,S}[0]$$
$$P_3, P_4 = E_{P,S}[P_1 \| P_2]$$
$$\cdots$$
$$P_{17}, P_{18} = E_{P,S}[P_{15} \| P_{16}]$$
$$S_{1,0}, S_{1,1} = E_{P,S}[P_{17} \| P_{18}]$$
$$\cdots$$
$$S_{4,254}, S_{4,255} = E_{P,S}[S_{4,252} \| S_{4,253}]$$

Where $E_{p,s}[Y]$ is the ciphertext produced by encrypting Y using Blowfish with the arrays S and P.

- A total of 521 executions of the Blowfish encryption algorithm are required to produce the final S- and P-arrays.
- Accordingly, Blowfish is not suitable for applications in which the secret key changes frequently. Further, for rapid execution, the P- and S-arrays can be stored rather than rederived from the key each time the algorithm is used.
- This requires over 4 kilobytes of memory. Thus, Blowfish is not appropriate for applications with limited memory, such as smart cards.

Encryption and Decryption

Blowfish uses two primitive operations:

- Addition: Addition of words, denoted by +, is performed modulo 2^{32}.
- Bitwise exclusive-OR: This operation is denoted by $\oplus$

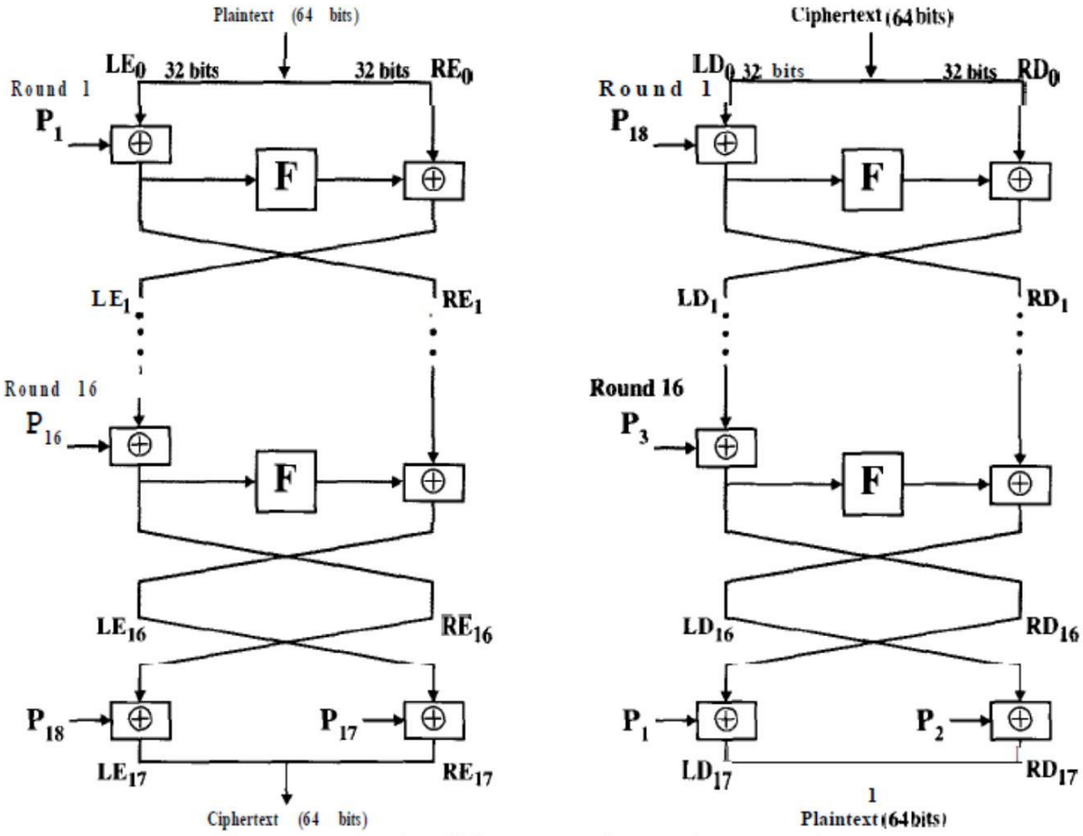


Figure Blowfish Encryption and Decryption.

In the above figure the encryption operation. The plaintext is divided into two 32-bit halves LE, and RE,. We use the variables LE, and RE, to refer to the left and right half of the data after round i has completed. The algorithm can be defined by the following pseudocode:

for $i = 1$ **to 16 do**
$RE_i = LE_{i-1} \oplus P_i$;
$LE_i = F[RE_i] \oplus RE_{i-1}$;
$LE_{17} = RE_{17} \oplus P_{18}$;
$RE_{17} = LE_{17} \oplus P_{17}$;

The function F is shown in below Figure. The 32-bit input to F is divided into 4 bytes. If we label those bytes a, b, c, and d, then the function can be defined as follows:

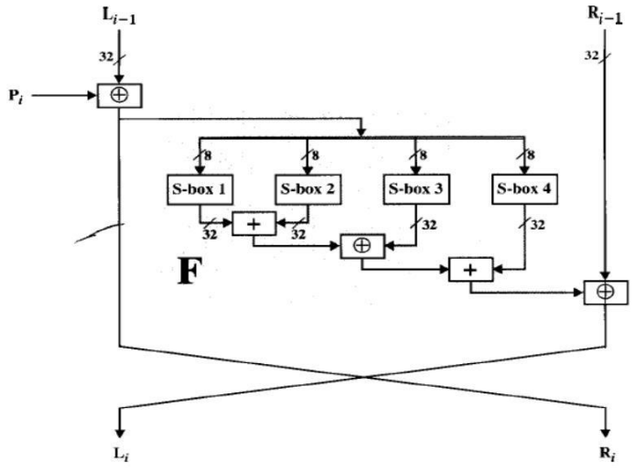


Figure Detail of Single Blowfish Round.

$$F[a, b, c, d,] = ((S_{1,a} + S_{2,b}) \oplus S_{3,c}) + S_{4,d}$$

Blowfish Decryption:

Blowfish decryption occurs in the same algorithmic direction as encryption. Rather than the reverse. The algorithm can be defined as follows:

for $i = 1$ to 16 do
$RD_i = LD_{i-1} \oplus P_{19-i}$;
$LD_i = F[RD_i] \oplus RD_{i-1}$;
$LD_{17} = RD_{17} \oplus P_1$;
$RD_{17} = LD_{17} \oplus P_2$;

Advantages or features of blowfish:

- A brute-force attack is even more difficult than may be apparent from the key length because of the time-consuming subkey-generation process. A total of 522 executions of the encryption algorithm are required to test a single key.
- The function F gives Blowfish the best possible avalanche affect for a Feistel network: In round i, every bit of L_{i-1}, affects every bit of R_{i-1}. In addition,every subkey bit is affected by every key bit. and therefore F has a perfect avalanche effect between the key (P,) and the right half of the data (R,) after every round.
- Every bit of the input to F is only used as input to one S-box. In contrast. In DES, many bits are used as inputs to two S-boxes. which strengthens the algorithm considerably against differential attacks. Schneier felt that this added complexity was not necessary with key-dependent S-boxes.
- Unlike in CAST, the function F in Blowfish is not round dependent. Schneier felt that such dependency did not add any cryptographic merit, given that the P-array substitution is already round dependent.

2.9. CAST-128:

It is an encryption algorithm.

It takes 64-bit plain text,128 bit key as input and produces 64-bit cipher text as output. It has 16 rounds.

Description of Algorithm:

CAST-128 belongs to the class of encryption algorithms known as Feistel ciphers; overall operation is thus similar to the Data Encryption Standard (DES). The full encryption algorithm is given in the following four steps.

INPUT: plaintext m1...m64;

key K = k1...k128.

OUTPUT: ciphertext c1...c64.

1. (key schedule) Compute 16 pairs of subkeys {Kmi, Kri} from K
2. Split the plaintext into left and right 32-bit halves L0 = m1...m32 and R0 = m33...m64.
3. It has 16 rounds for i from 1 to 16, compute Li and Ri as follows:

   Li = Ri-1;
   Ri = Li-1 ⊕ f(Ri-1,Kmi,Kri), where f is function (f is of Type 1, Type 2, or Type 3, depending on i).
4. c1...c64 <-- (R16,L16). (Exchange final blocks L16, R16 and concatenate to form the ciphertext.)

Decryption is identical to the encryption algorithm given above, except that the rounds (and therefore the subkey pairs) are used in reverse order to compute (L0,R0) from (R16,L16).

Pairs of Round Keys:

CAST-128 uses a pair of subkeys per round: a 32-bit quantity “Km” is used as a "masking" key and a 5-bit quantity “Kr” is used as a "rotation" key.

Non-Identical Rounds:

Three different round functions are used in CAST-128.

The rounds are as follows

- where "D" is the data input to the f function and "Ia" - "Id" are the most significant byte through least significant byte of I, respectively).
- All functions use the operation "+" and "-" are addition and subtraction ⊕ XOR, and "<<<" is the circular left-shift operation.

Type 1: I = ((Kmi + D) <<< Kri)

f = ((S1[Ia] ⊕ S2[Ib]) - S3[Ic]) + S4[Id]

Type 2: I = ((Kmi ⊕ D) <<< Kri)

f = ((S1[Ia] - S2[Ib]) + S3[Ic]) ⊕ S4[Id]

Type 3: I = ((Kmi - D) <<< Kri)

f = ((S1[Ia] + S2[Ib]) ⊕ S3[Ic]) - S4[Id]

Rounds 1, 4, 7, 10, 13, and 16 use f function Type 1.
Rounds 2, 5, 8, 11, and 14 use f function Type 2.
Rounds 3, 6, 9, 12, and 15 use f function Type 3.

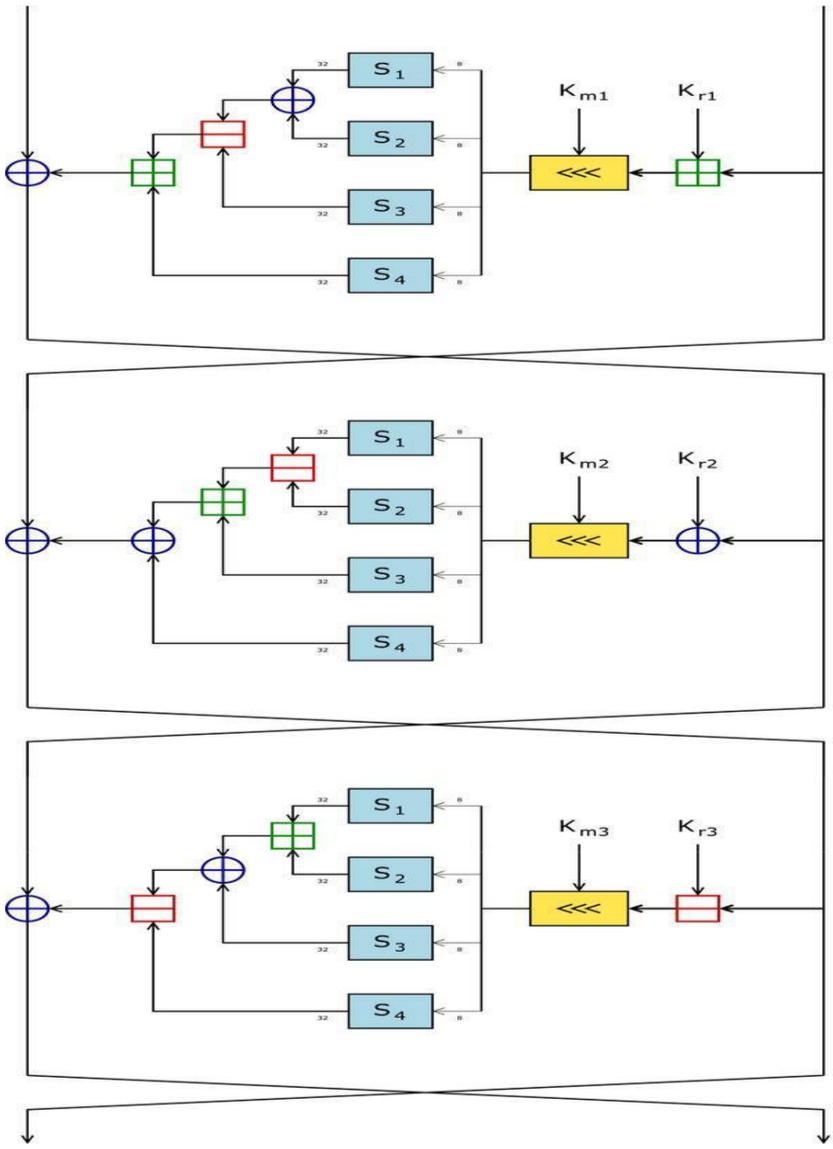


Figure: Three rounds of the CAST-128 block cipher

Substitution Boxes:

- CAST-128 uses eight substitution boxes:

  s-boxes S1, S2, S3, and S4 are round function s-boxes; S5, S6, S7, and S8 are key schedule s-boxes.

Masking Subkeys and Rotate Subkeys:

Let Km1, ..., Km16 be 32-bit masking subkeys (one per round).

Let Kr1,…., Kr16 be 32-bit rotate subkeys (one per round); only the least significant 5 bits are used in each round.

for (i=1; i<=16; i++)

{ Kmi = Ki; Kri

= K16+i; }

2.10. INTERNATIONAL DATA ENCRYPTION ALGORITHM(IDEA):

- IDEA (International Data Encryption Algorithm) was originally called IPES (Improved Proposed Encryption Standard).
- It was developed by Xuejia Lai and James L. Massey of ETH Zuria.
- IDEA was designed to be efficient to compute in software. It encrypts a 64-bit block of plaintext into a 64-bit block of ciphertext using a 128-bit key.
- It was published in 1991, so cryptanalysts have had time to find weaknesses.
- IDEA is similar to DES in some ways.
- Both of them operate in rounds, and both have a complicated mangler function that does not have to be reversible in order for decryption to work.
- Instead, the mangler function is run in the same direction for encryption as decryption, in both IDEA and DES.
- In fact, both DES and IDEA have the property that encryption and decryption are identical except for key expansion.
- With DES, the same keys are used in the reverse order
- with IDEA, the encryption and decryption keys are related in a more complex manner.

Primitive Operations

- Each primitive operation in IDEA maps two 16-bit quantities into a 16-bit quantity.
- IDEA uses three operations
- ⊕-XOR
- +-Adddition all easy to compute in software, to create a mapping.

⊗ Multiplication Operation.

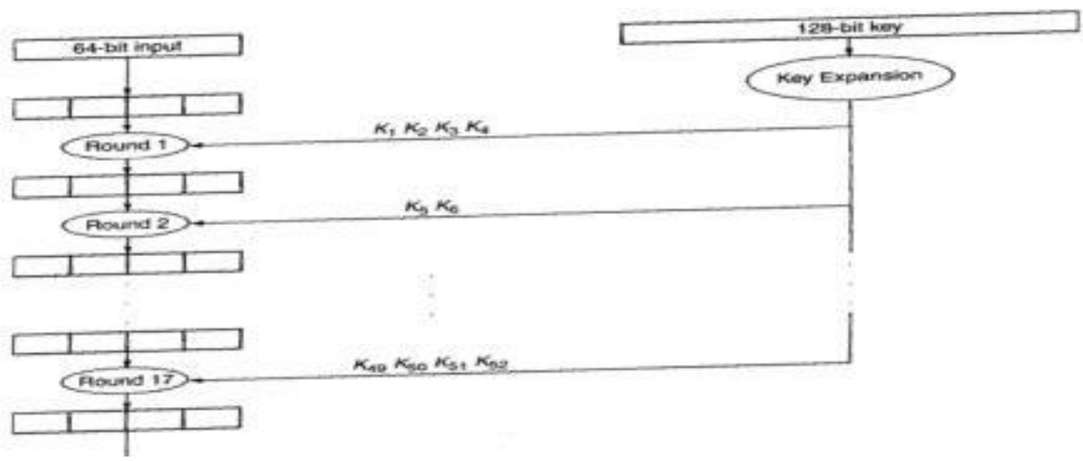


Figure: basic structure of IDEA

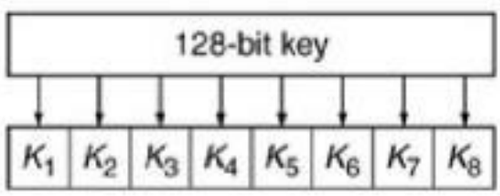
It has total 17 rounds

In IDEA, Odd rounds accepts 4 subkeys.

Even rounds accepts 2 subkeys.

Key expansion:

- The 128-bit key is expanded into 52 16-bit keys, K_1, K_2,...K_{52}.
- The key expansion is done differently for encryption than for decryption.
- Once the 52 keys are generated, the encryption and decryption operations are the same.
- The 52 encryption keys are generated by writing out the 128-bit key and, starting from the left, chopping off 16 bits at a time.
- This generates eight 16-bit keys



One Round:

- It has 17rounds, where the odd numbered rounds are different from the even numbered rounds.
- Each round takes the input a 64-bit quantity and treats it as four 16-bit quantities X_a,X_b,X_c,X_d. Mathematical Operations are performed on it.
- In IDEA, Odd rounds accepts 4 subkeys.
- Even rounds accept 2 subkeys.

An odd round, therefore has as input X_a, X_b, X_c, X_d and keys K_a, K_b, K_c, K_d. An even round has as input X_a, X_b, X_c, X_d and keys K_e and K_f.

Odd round:

The odd round is simple. X_a is replaced by $X_a \otimes K_a$. X_d is replaced by $X_d \otimes K_d$. X_c is replaced by $X_b + K_b$. X_b is replaced by $X_c + K_c$.

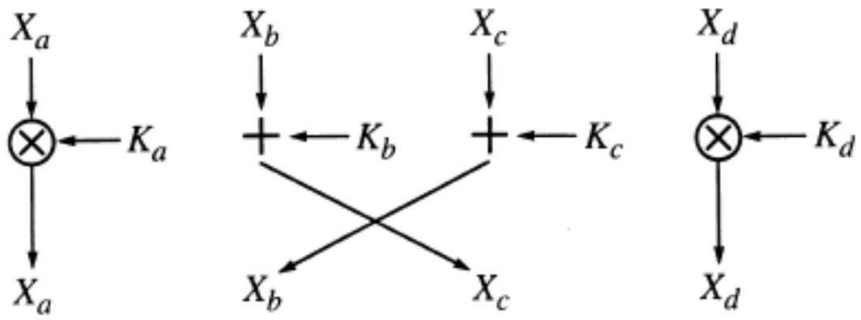


Figure: IDEA odd round

Even Round:

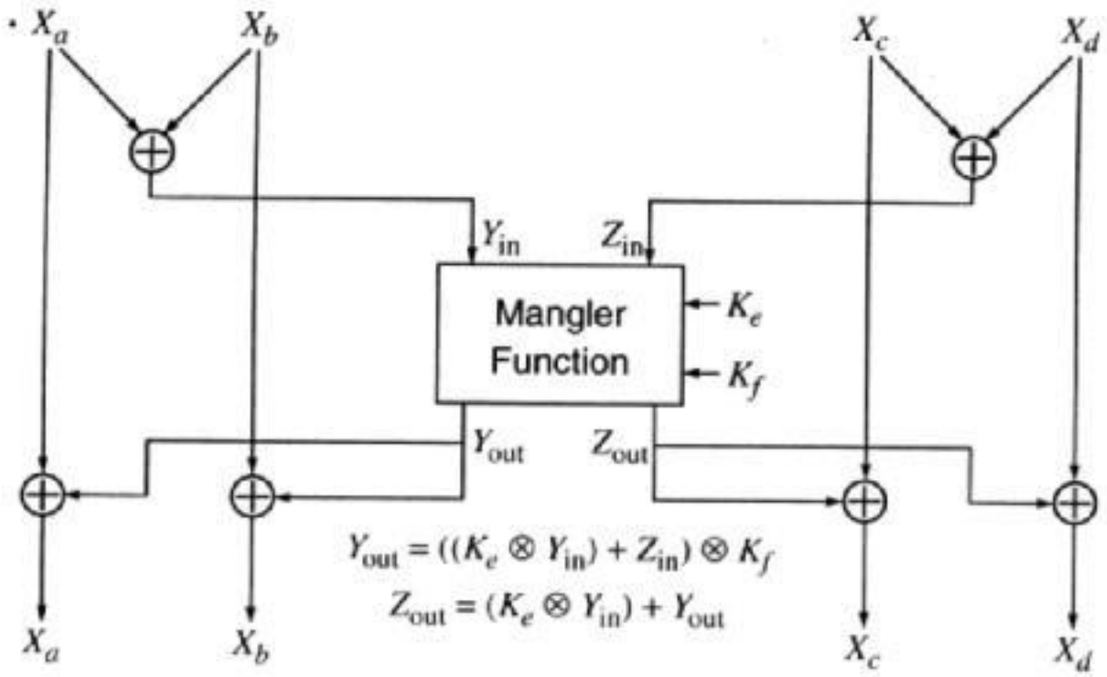


Figure: IDEA Even Round

The even round is a little more complicated. Again, we have X_a, X_b, X_c, and X_d. We have two keys, K_e and K_f. We're going to first compute two values, which we'll call Y_{in} and Z_{in}. We'll do a function, which we'll call the *mangler function*, which takes as input Y_{in}, Z_{in}, K_e, and K_f and produces what we'll call Y_{out} and Z_{out}. We'll use Y_{out} and Z_{out} to modify X_a, X_b, X_c, and X_d.

$$Y_{in} = X_a \oplus X_b \quad Z_{in} = X_c \oplus X_d$$

$$Y_{out} = ((K_e \otimes Y_{in}) + Z_{in}) \otimes K_f$$
$$Z_{out} = (K_e \otimes Y_{in}) + Y_{out}$$

2.11. Block Cipher Modes of Operation:

A block cipher algorithm is a basic building block for providing data security. To apply a block cipher in a variety of applications, different "modes of operation" have been defined by NIST. In essence, a mode of operation is a technique for enhancing the effect of a cryptographic algorithm or adapting the algorithm for an application, such as applying a block cipher to a sequence of data blocks or a data stream. The modes are intended to cover virtually all the possible applications of encryption for which a block cipher could be used.

Electronic Codebook Mode:

The simplest mode is the electronic codebook (ECB) mode, in which plaintext is handled one block at a time and each block of plaintext is encrypted using the same key (Figure a & b). The term codebook is used because, for a given key, there is a unique ciphertext for every b-bit block of plaintext. For a message longer than b bits, the procedure is simply to break the message into b-bit blocks, padding the last block if necessary. Decryption is performed one block at a time, always using the same key. In Figure, the plaintext (padded as necessary) consists of a sequence of b-bit blocks, P1, P2,..., PN; the corresponding sequence of ciphertext blocks is C1, C2,..., CN.

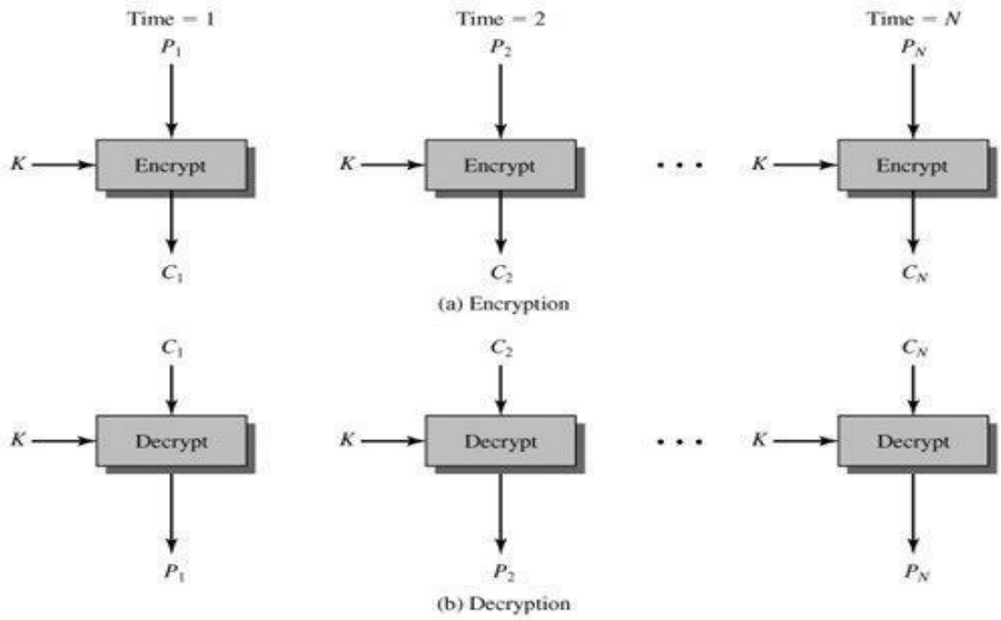


Figure. Electronic Codebook (ECB) Mode

The ECB method is ideal for a short amount of data, such as an encryption key. Thus, if you want to transmit a DES key securely, ECB is the appropriate mode to use. The most significant characteristic of ECB is that the same b-bit block of plaintext, if it appears more than once in the message, always produces the same ciphertext.

For lengthy messages, the ECB mode may not be secure. If the message is highly structured, it may be possible for a cryptanalyst to exploit these regularities. For example, if it is known that the message always starts out with certain predefined fields, then the cryptanalyst may have a number of known plaintext-ciphertext pairs to work with. If the message has repetitive elements, with a period of repetition a multiple of b bits, then these elements can be identified by the analyst. This may help in the analysis or may provide an opportunity for substituting or rearranging blocks.

Cipher Block Chaining Mode:

To overcome the security deficiencies of ECB, we would like a technique in which the same plaintext block, if repeated, produces different ciphertext blocks. A simple way to satisfy this requirement is the cipher block chaining (CBC) mode.

In this scheme, the input to the encryption algorithm is the XOR of the current plaintext block and the preceding ciphertext block; the same key is used for each block. In effect, we have chained together the processing of the sequence of plaintext blocks. The input to the encryption function for each plaintext block bears no fixed relationship to the plaintext block. Therefore, repeating patterns of *b bits are not exposed.*

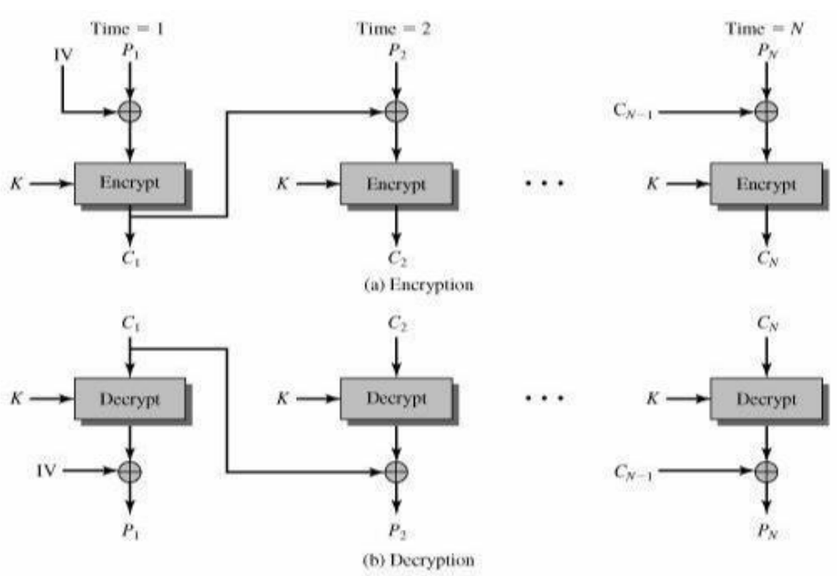


Figure : Cipher Block Chaining (CBC) Mode

For decryption, each cipher block is passed through the decryption algorithm. The result is XORe d with the preceding ciphertext block to produce the plaintext block. To produce the first block of ciphertext, an initialization vector (IV) is XORed with the first block of plaintext. On decryption, the IV is XORed with the output of the decryption algorithm to recover the first block of plaintext. The IV is a data block that is that same size as the cipher block.The IV must be known to both the sender and receiver but be unpredictable by a third party. For maximum security, the IV should be protected against unauthorized changes. This could be done by sending the IV using ECB encryption. Because of the chaining mechanism of CBC, it is an appropriate mode for encrypting messages of length greater than *b bits.* CBC mode can be used for authentication.

Cipher Feedback Mode:

The DES scheme is essentially a block cipher technique that uses b-bit blocks. However, it is possible to convert DES into a stream cipher, using either the cipher feedback (CFB) or the output feedback mode. Figure depicts the CFB scheme. In the figure, it is assumed that the unit of transmission is s bits; a common value is s = 8. As with CBC, the units of plaintext are chained together, so that the ciphertext of any plaintext unit is a function of all the preceding plaintext. In this case, rather than units of b bits, the plaintext is divided into segments of s bits.

First, consider **encryption**. The input to the encryption function is a b-bit shift register that is initially set to some initialization vector (IV). The leftmost (most significant) s bits of the output of the encryption function are XORed with the first segment of plaintext P1 to produce the first unit of ciphertext C, which is then transmitted. In addition, the contents of the shift register are shifted left by s bits and C is placed in the rightmost (least significant) s bits of the shift register. This process continues until all plaintext units have been encrypted.

For decryption, the same scheme is used, except that the received ciphertext unit is XORed with the output of the encryption function to produce the plaintext unit.

Let $S_S(X)$ be defined as the most significant s bits of X. Then

$C1 = P1 \oplus S_S[E(K,IV)]$

Therefore,

$P1 = C1 \oplus S_S [E(K, IV)]$

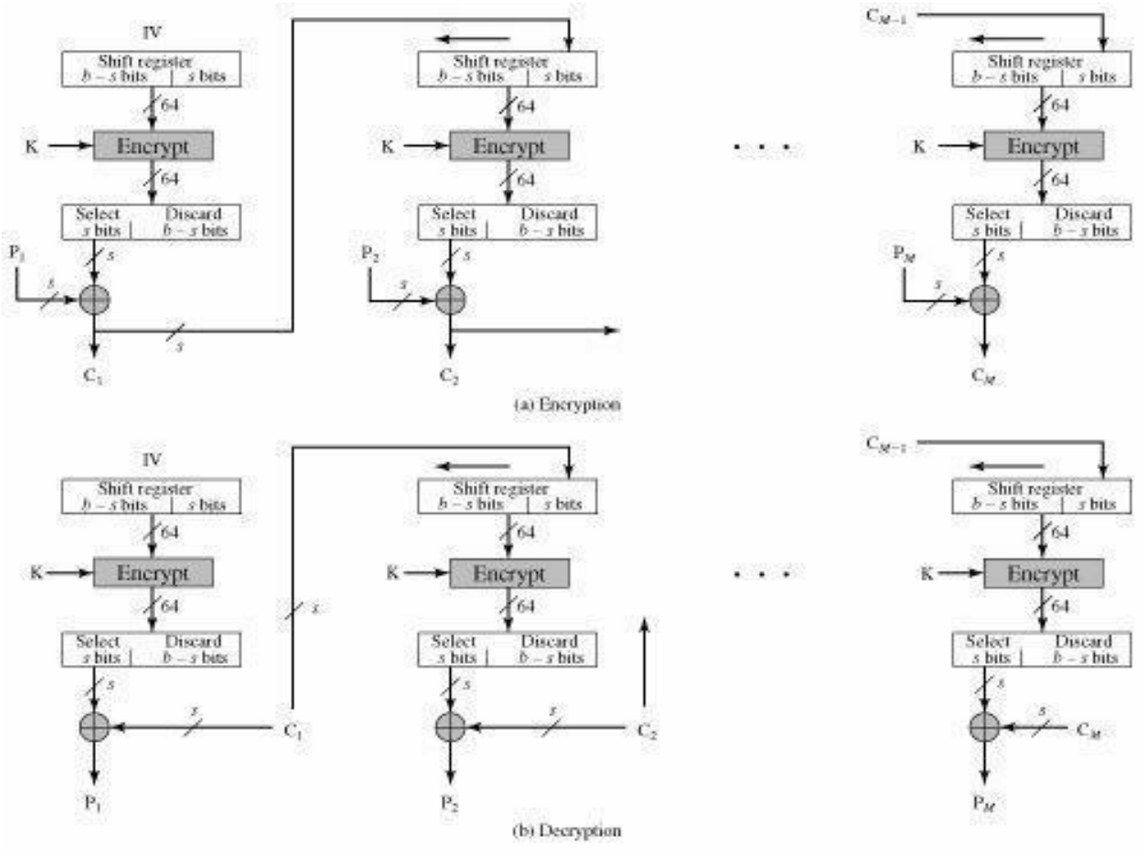


Figure. s-bit Cipher Feedback (CFB) Mode

Output Feedback Mode:

The output feedback (OFB) mode is similar in structure to that of CFB, as illustrated in Figure. As can be seen, it is the output of the encryption function that is fed back to the shift register in OFB, whereas in CFB the ciphertext unit is fed back to the shift register. One advantage of the OFB method is that bit errors in transmission do not propagate. For example, if a bit error occurs in C_1 only the recovered value of is P_1 affected; subsequent plaintext units are not corrupted. With CFB, C_1 also serves as input to the shift register and therefore causes additional corruption downstream. The disadvantage of OFB is that it is more vulnerable to a message stream modification attack than is CFB.

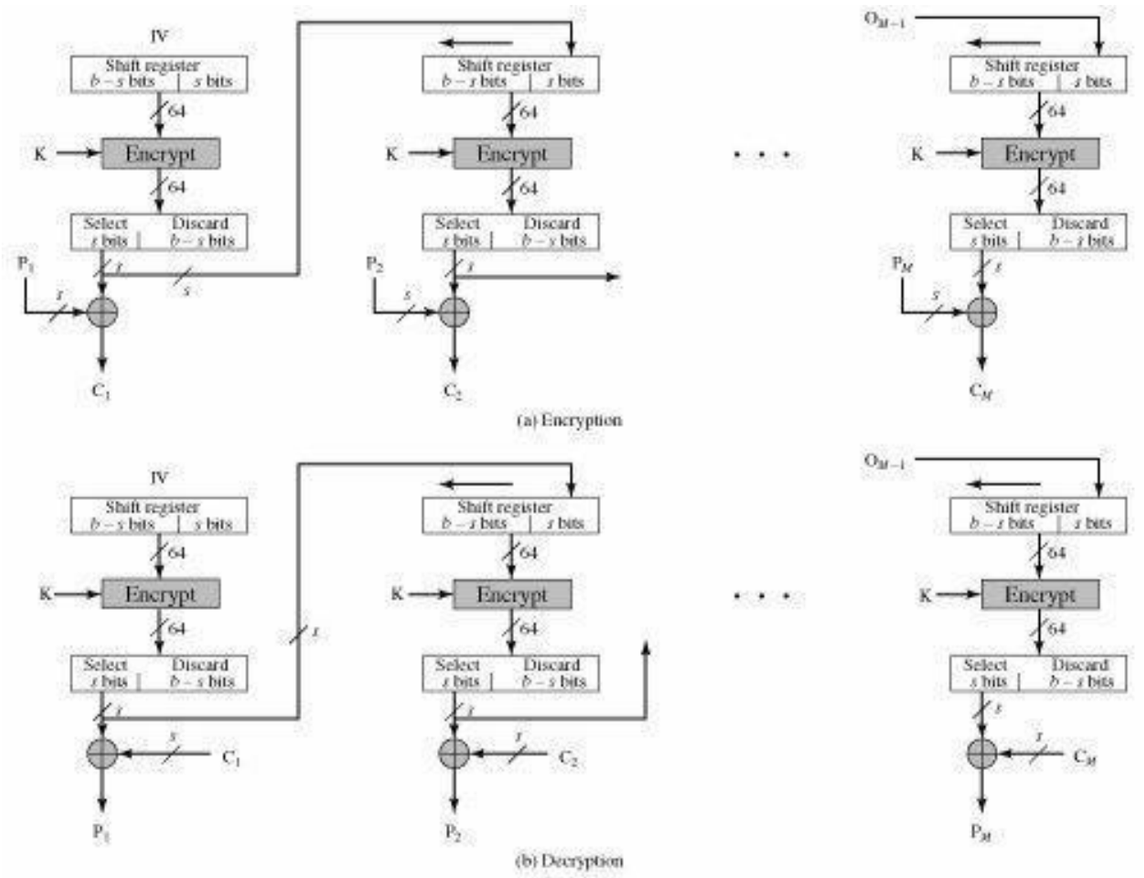


Figure. s-bit Output Feedback (OFB) Mode

Counter Mode:

In CTR mode a counter, equal to the plaintext block size is used. The only requirement is that the counter value must be different for each plaintext block that is encrypted. Typically, the counter is initialized to some value and then incremented by 1 for each subsequent block. For encryption, the counter is encrypted and then XORed with the plaintext block to produce the ciphertext block; there is no chaining. For decryption, the same sequence of counter values is used, with each encrypted counter XORed with a ciphertext block to recover the corresponding plaintext block.

Advantages:

1. Hardware efficiency
2. Software efficiency
3. Preprocessing
4. Random access
5. Provable security
6. Simplicity

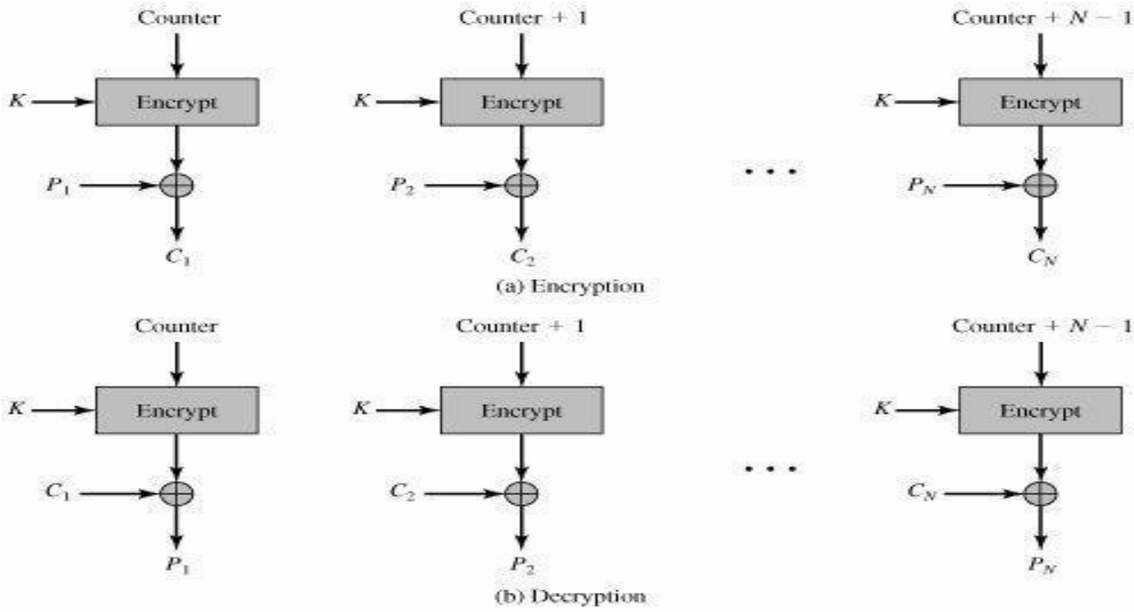


Figure. Counter (CTR) Mode